Enabling Dynamic Communication for Runtime Circuit Relocation

Enabling Dynamic Communication for Runtime Circuit Relocation

Adewale Adetomi, Godwin Enemali, and Tughrul Arslan, *Senior Member*

***Abstract*—Runtime circuit relocation has been proposed for mitigating the effect of permanent damages in reconfigurable hardware like FPGAs with potentials to improve reliability and reduce or eliminate system downtime. However, a major obstacle to the adoption of circuit relocation is the presence of static communication links between circuits. Existing solutions to this are either computationally expensive or counter-intuitive to system reliability. This article proposes a dynamic communication mechanism that is able to circumvent the static links. The clock buffers in a typical FPGA use independent wires and thus, do not constitute static routing. These are repurposed as network links to provide dynamic communication for relocatable circuits, with a demonstrator based on a 4-node star network showing a bandwidth of 428.58 Mbps for a 32-bit payload at an overhead of only 144 slices on the Artix-7 FPGA.**

***Index Terms*—circuit relocation, network on chip, reconfigurable computing, reliability**

I. Introduction

THE use of reconfigurable hardware like FPGAs for computing has become more popular recently. In particular, the possibility of modifying the configured circuit in runtime is attractive for error mitigation in harsh environments like space, where radiations can induce temporary errors and permanent damages. One key technique that has been proposed for mitigating permanent damages is circuit relocation, which involves moving a configured circuit (or *task*) from one place to the other on the chip [1]. In addition, circuit relocation is important in *Wear-Levelling* (WL) approaches for mitigating ageing-induced permanent damages. In general, WL is any strategy deployed to pre-empt damages before they occur, spreading out wear and prolonging the lifespan of the chip. WL can involve the alternate usage of chip resources over time to achieve uniform ageing [2][3].

Runtime circuit relocation is enabled by the *Dynamic Partial Reconfiguration* (DPR) technology in FPGAs [4], which allows a part of the FPGA to be reconfigured while the rest of the chip remain functional. In the *Partial Reconfiguration* (PR) flow, the chip area is divided into a static region (for non-PR circuits) and a reconfigurable region, which is floor-planned into multiple *Reconfigurable Partitions* (RPs) for *Reconfigurable Modules* (RMs) to be (re)configured. In general, an RM can only be placed in an RP for which it has been floor-planned at compile time. Circuit relocation allows an RM to be used in un-associated RPs.

An important challenge with circuit relocation, which has limited its applicability, is how to provide dynamic communication for relocated circuits in runtime since inter-circuit links are statically determined at compile time. Runtime routing is a possible solution to dynamic communication but it is both complicated and computationally expensive, often requiring several thousands of clock cycles [5]. In addition, there can be static routes in the target location which belong to the static region, and these must be preserved.

Network on Chip (NoC) has come to be regarded as the future of on-chip communication, owing to advantages such as modularity and concurrency [6]. However, NoCs typically use the general routing resources of the chip as network links, thereby constituting static routes which are a barrier to relocation. To alleviate this problem, a network link that does not use the general interconnect resources should be used where possible. Incidentally, it turns out that most FPGA-based designs do not use the available on-chip global and horizontal clock buffers [7] and invariably, the clock network. Repurposing these buffers and networks for use as communication network links would help circumvent the restriction of static routes and allow the arbitrary relocation of circuits. Since the clock buffers do not use the general logic routing resources [7], the path from a transmitting circuit to a receiving circuit is free of logic interconnections. Moreover, these otherwise redundant resources, which are not used for their intended clocking-related purposes have already been paid for in silicon. As such, using them for communication-related purposes represents an added value.

Furthermore, it is important to note that routing congestions are often the reason static routes cross into RPs from the static region. A way of reducing routing congestion, especially at the interface of circuits, and thus, reducing the number of static routes is to use bit-serial interconnections between circuits, as this has been shown to have reduced footprint and congestion factors [8]. Incidentally, our use of clock buffers for communication calls for the adoption of bit-serial connection at the clock buffer level. However, multiple bit-serial connections can be used depending on the availability of clock buffers in the target FPGA. In addition, a bit-serial implementation is beneficial because it helps in easily meeting the requirement for the preservation of existing static routes, while at the same time garnering the other benefits of bit-serial over bit-parallel interconnects, which include high speed and power savings as demonstrated in [8] and [9].

Because the proposed technique incurs a low overhead of resources and involves the unique use of clock buffers for serial network interconnection, we have termed it *Clock-Enabled Low-Overhead Communication* (CELOC). Without loss of generality of application, CELOC has been targeted at Xilinx FPGAs and demonstrated on the 7 Series fabric.

A. Adetomi, G. Enemali and T. Arslan are with the Institute for Integrated Micro and Nano Systems, University of Edinburgh, King's Buildings, Edinburgh EH9 3FF, Scotland, United Kingdom. (e-mail: {a.adetomi, g.enemali, t.arslan}@ ed.ac.uk).

The main motivation for proposing CELOC is to circumvent RP-crossing static routes which prevent the relocation of circuits in a PR design. We name two type: *Type A* – those that originate from (static logic blocks that belong to) the static design and are allowed to cross into RPs by the design tool [4]; and *Type B* – those that derive from inter-circuit communication via LUTs and registers. Every other form of routing, including those that may go through clock buffers are internal to RPs and are not bottlenecks to relocation as they are part of the RM. *Type-A* static routes can be easily mitigated by enlarging offending partitions. Our aim is to address *Type-B* static routes by preventing inter-circuit communication from going through the general interconnect when crossing RPs.

We have presented preliminary results in [10], [11], and [12]. This article provides more in-depth descriptions and reports on further findings. The key contributions include:

1) The use of the clocking infrastructures of an FPGA for on-chip dynamic communication.
2) Characterization of different clock buffer combinations for communication.
3) A prototype NoC that uses CELOC to support runtime circuit relocation.

II. Background

On-chip communication architectures can be grouped into three main categories, namely P2P, Bus, and NoC, based on the structure of the physical interconnect, the protocol of data transfer, and the interface design [13]. The main characteristic of P2P interconnect is the simple and direct interconnection between two communicating circuits, but it is quite inefficient in terms of scalability as the number of cores increases. The shortcomings of P2P architecture scales up in shared buses. While shared bus allows multiple cores to communicate by granting them access to a central global bus; however, because of the diverse nature and the sheer number of these cores, buses become longer, introducing longer communication latencies, and consuming more power [14]. Buses are not flexible enough as an addition of a new module requires that the entire system be redesigned. As a result, NoCs have been proposed as the future of on-chip communication.

A. Network-on-Chip for Communication

The NoC was borne out of the need to improve scalability, modularity, and performance among other factors, in on-chip communication [15]. This need arose because of the increase in the number and type of modules or processing elements running and communicating on a device. CPUs, graphics processors, DSPs, memory elements, and other modules with different functionalities became common-place on a single chip, effectively giving rise to the idea of System-on-Chip.

The deficiencies of dedicated P2P and bus architectures are rooted in the reliance on the routing of wires between communicating circuits. With the increase in networking requirement as more cores are added, wires become long and connections more complicated, leading to increased power consumption. It is clear that modern on-chip communication cannot rely on connection-based interconnections. These deficiencies have given rise to the notion of routing packets, and not wires [6], which is the main idea behind the NoC. Instead of establishing P2P connections, whether based on direct dedicated interconnections or shared buses, the NoC abstracts the Data Link Layer (data transfer on wired links) from the Application/Presentation Layer (the on-chip cores). That is, it decouples computation from communication with the potential to bring about unprecedented levels of scalability and performance. The general structure of an NoC is shown in Fig. 1, with 3-by-3 nodes as an example. NoCs come in various forms targeted at addressing different performance metrics, but in general, they are made up of routers, adapters, and links that connect all the cores (processing elements or circuits) on a chip. Each core is interfaced to the network via a network adapter that implements a network interface on the network side and a core interface on the core side.

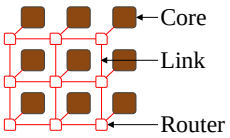

Fig. 1: The architecture of a generic NoC

There are several architectural features of NoCs. However, since the concept of NoCs is an already comprehensively covered subject, more extensive details on its basics can be found in [15] and [16]. Nevertheless, it is pertinent to identify the performance parameters of NoCs, which are bandwidth, throughput, and latency [16]. Measured in bits per second (bps), the bandwidth of an NoC is the maximum rate of data transfer and it usually considers the entire packet. Throughput makes allowance for the fact that a packet usually contains non-message-related header and tail information. As such, it measures the rate of transfer of the message payload in messages per second or messages per clock cycle. Both bandwidth and throughput scale with the number of channels. Latency is the time elapsed from the instance a packet departs a source node to being completely received at the destination.

B. Shortcomings of NoCs

As promising as NoCs are, they have their downsides. Though they offer a good communication solution when compared to dedicated P2P and bus communications, there is an attendant resource overhead, which can be significant in smaller devices. That is, NoCs lead to an increase in the footprint of the overall design, and this is due mainly to the additional resources used for the routers to grant network access to the tasks. Depending on the size of the network, an overhead of up to 34.8% (3227 slices) for a 2-by-2 network is not impossible [17].

Moreover, while compared to shared buses, NoCs lend themselves more readily to runtime circuit placement because of their support for easy modularity and scalability; however, the static routes of the network links still constitute a bottleneck to circuit relocation. In particular, the traditional NoC links pose the challenge of static routes as these links are constructed from the chip's general routing resources and are free to cross the reconfigurable partitions in partially reconfigurable system architecture. In a bit-parallel NoC, the network adapter at each node creates static routes that cross into other nodes. A bit-serial NoC that uses general interconnects as links would have lesser static routes, but it

would still require online redetermination of route in order to support dynamic communication.

Meanwhile, it is possible to completely do away with routers and still have comparable or better network performance as demonstrated in [18], where a routerless NoC shows a 7.7x reduction in power, a 3.3x reduction in area, a 1.3x reduction in zero-load packet latency, and a 1.6x increase in throughput when compared to a router-based NoC.

C. Bit-Serial and Bit-Parallel NoCs

The interconnections that carry packets from router to router can be made up of several single wires to form a parallel link that is able to switch a multi-bit data at a time. This is the typical case with NoCs and such NoCs can be referred to as bit-parallel NoCs. On the other hand, the link can also be composed of a single wire which is able to transmit a bit of data at a time, and as such, the resulting NoC can be termed bit-serial.

Although bit-parallel NoCs generally offer higher throughputs, however, it has been shown that a serial implementation has the potential to reduce the area overhead and power utilization of NoCs [9] while at the same time improving noise and signal interference, offering simpler network layout, and enhancing timing verification. It turns out that because of the efficiency it brings, high-speed serial communication is the current trend in digital design, e.g., PCI Express. As a result of the serial single-wire implementation, the usual performance-limiting skew on parallel links is localized to a single link and as such a much higher frequency is possible with a serial link.

A bit-parallel link can provide a higher throughput than a bit-serial one when clocked at the same frequency. However, in the long run, a bit-serial link can achieve higher throughput if it can be clocked at a fast enough rate, at which point a bit-parallel link fails because of skew. For instance, in [8], the authors demonstrate bit-serial NoC routers that are 2-3x faster than their equivalent bit-parallel routers even with some level of pipeline optimization in the parallel implementation.

To reduce an NoC's area utilization, bit-serial network access can be used as proven in [9], where in a comparative analysis of serial and parallel interconnects, the authors note significant improvements of up to 5.5× and 17× power consumption and area utilization respectively of serial links over parallel links. Similarly, in [8], the author observes that bit-parallel routers are 8× (for LUTs) and 23× (for FFs) larger than bit-serial routers. In addition, bit-serial designs are noted to have route congestion factors of only 1-2% compared to 10-20% for their bit-parallel counterparts.

Incidentally, because of their limited number in a typical FPGA, the clock buffers are better suited as bit-serial links, with each clock buffer able to drive a bit from the source circuit to the destination circuit in one clock cycle.

D. The Need for Dynamic Communication

One of the key requirements for circuit relocation is the provision of dynamic communication for relocatable circuits. As such, the need for dynamic communication infrastructures is a salient one. An approach to dynamic communication is taken in DyNoC, a dynamic network-on-chip architecture [19]. While several research works have been carried out on dynamic or reconfigurable NoCs, most do not actually consider the placement of a new task. Rather, they are mostly concerned with the runtime restructuring of the network topology or packet routing to meet changing communication needs as seen in ReNoC [20] and Hoplite [21] respectively. On the other hand, DyNoC's approach to dynamic communication involves placing a new circuit over existing deactivated network routers while leaving surrounding routers free for communication. With this arrangement, a new circuit can be placed anywhere on the mesh network with continued access to the network. However, we deem this approach to still have the challenges of static routes as the authors do not seem to have provided details on how these are managed and their implementation diagram [19] shows routings crisscrossing the entire floorplan. Indeed, it is unlikely that the authors intend DyNoC to be a communication network for relocatable circuits, as this is not a claim in the work.

An ideal situation for dynamic communication is to have no static interconnects to deal with or need to create routes on the fly. A step in that direction is taken in [22], where the authors present a communication mechanism that involves using the *Internal Configuration Access Port* (ICAP) of an FPGA to transfer data between arbitrarily-placed hardware tasks, in the context of achieving the relocatability of tasks. This is done by connecting memory elements (distributed RAMs or BRAMs) to the inputs and outputs of circuits to serve as data memories and using the ICAP as a side channel to copy data from output memories to input memories thereby avoiding static interconnects. The data contents of a memory element can be accessed online from the device's configuration memory.

This idea of moving data from one task to another without using physical wires can be seen as a form of relocation and it has limitations and consequences as highlighted in [23]. There is no way to know when a task has finished computation apart from polling the task. With multiple tasks possibly simultaneously active, this is even more demanding. There are three operations needed to be performed for each data relocation – polling, readback, and writing. All these operations have to be serialized since the ICAP is a single resource. That is, ICAP-based data relocation is not concurrent and this is the main bottleneck with it.

Furthermore, the single nature of the ICAP could have an implication on system reliability and performance. The ICAP has a maximum theoretical bandwidth of 400 MB/s [4] and Xilinx recommends that more than 99% of this bandwidth should be dedicated to *Soft Error Mitigatio*n (SEM) [24] for the entire device. Using at least 99% for SEM means that only 4 MB/s of the ICAP's bandwidth is available for other functions. With communication drawn in, there are two system functions competing for the remaining 4 MB/s. In other words, time spent on communication is time not available for SEM and configuration.

III. Clocking Resources in the Xilinx 7 Series FPGA and the Features Exploited by CELOC

An understanding of the types and features of the clock buffers available in the FPGA is crucial in the design and implementation of CELOC. A typical Xilinx FPGA is divided into areas called clock regions [7] containing configurable logic blocks, block RAMs, and DSPs. Different networks of

clock buffers feed clock signals to these resources, with the clock network of all modern FPGAs based on the Spine-and-Ribs topology [25], where vertical spines drive clock signals into horizontal ribs. Eventually, local ribs in the clock regions clock logic resources directly. Fig. 2 shows the clock network distribution and interaction in a single clock region of the 7 series FPGA. There is a horizontal clock row (HROW) that spans the entire length of the clock region. Clock signals switch vertically upward and downward from the HROW to reach logic resources. The most important thing to note and that which is being exploited in our adaptation of clock buffers and networks for inter-task communication is that the clock networks use independent physical wires different from the general logic interconnects. Inside a clock region, switch matrices route clock signals to the logic resources.

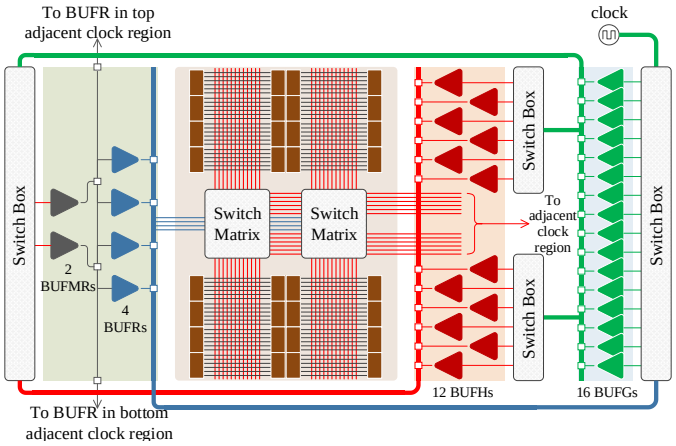

Fig. 2: The network distribution in the clock region of a 7 series FPGA

In order to realize CELOC, two factors are important for a clock buffer – the span or reach of the buffer, and the availability of a CE pin for logic functions. The former determines how far on the chip a communication signal can travel while the latter affects the number of transmitting nodes a CELOC network can support. In addition, the clock buffer must be user-accessible in the design tool, that is, it must be possible to instantiate and place it in order to control connections to and from it. For instance, in the UltraScale architecture, BUFCE_LEAF clock buffers are not user-accessible and are for routing clocks vertically from horizontal distribution [26]. However, this does not have a limiting effect on CELOC as this routing is guaranteed to be non-RP-crossing [4]. Furthermore, any other use of a clock buffer is acceptable so far it is internal to the RM being relocated, as is the case with the BUFCE_LEAFs.

A. *Global Clock Buffers/Multiplexers – BUFG(CE)*

The global clock buffers drive the global vertical clocking backbone in the device. Their reach spans the entire FPGA and they can feed any clocking point in the device. As such, they can be used for device-wide communication. The global clock nets have the capacity to drive not only CLK inputs of logic resources but also the Set/Reset (SR) and CE inputs of registers. This feature is particularly important in achieving CELOC-based dynamic communication, as it allows a communication clock signal to be received via the SR input of a register, ensuring that no local (static) route crosses the RP.

B. *Horizontal Clock Buffers – BUFH and BUFHCE*

They drive the horizontal global clock tree spines but span only two horizontally-adjacent regions. The CE of the BUFHCE can be used to achieve a true logic function on a clock cycle-to-cycle basis, allowing the control of the transfer of the clock input to the output of the buffer. The BUFHCE can be used for horizontal communication in a CELOC-based NoC. There are 12 BUFH(CE)s in each region.

C. *Multi-Region Clock Buffers – BUFMR and BUFMRCE*

These are used to enable multi-region clocking by directly driving regional clock and I/O buffers (BUFRs and BUFIOs) in the same clock region and the ones above and below it. Like the BUFHCE, their CE can be used to control the input-to-output transfer. The BUFMRCE can be used to achieve a CELOC-based network that is local to three vertical clock regions. There are two BUFMR(CE)s in each clock region.

D. *Regional Clock Buffers - BUFR*

These can drive any clocking point drivable by a global clock in a single clock region. In each region, there are four clock trees and nets, which are distinct from the global ones. There are four BUFRs driving these independent trees and nets in each region, allowing multiple unique clocks to feed a single design. These buffers can be used in both BUFR and BUFRCE configurations. They have two control lines, the CE and the clear (CLR), which can only be used in the frequency division mode. That is, the CE can only be toggled if the BUFR_DIVIDE option is set to any number other than "BYPASS" when the buffer is instantiated in RTL. With regards to CELOC, BUFR(CE)s can be used for intra-region communication and are essential for transferring signals out of a clock region as they are able to connect directly to BUFGs.

IV. Adaptation of Clock Network Infrastructures for On-Chip Communication

The availability of a diverse range of clock buffers with global and local spans in the Xilinx FPGAs offers a unique possibility that can be utilized to achieve on-chip communication functionality. CELOC involves an adaptation of these clock buffers to serve as binary ('0' or '1') signal transmitters and receivers on the FPGA. Meanwhile, would repurposing clock buffers for communication not be detrimental to their intended functionality? While the clock buffers and nets are precious and are available in the chip predominantly for clocking-related functionalities like glitchless multiplexing between clock sources, clock gating to reduce dynamic power consumption, and elimination of clock distribution delays, however, most FPGA designs contain several unused clock buffers [7]. With CELOC, these redundant buffers are repurposed to provide a static-route-free inter-communication for relocatable circuits.

Fig. 3 presents the CELOC concept in a diagrammatic form. By gating a free-running communication clock using a clock buffer, it is possible to send data from a transmitting (TX) task to a receiving (RX) task from any location on the device to another reachable by the buffer. At the TX end, a Serializer works in a *Parallel-In Serial-Out* (PISO) version to send data while a Deserializer at the RX reverses the operation in a *Serial-In Parallel-Out* (SIPO) version. CELOC requires

an RX task to be fed with three clocks: task_clock, com_clock, and data_clock. The task_clock is used to clock tasks while com_clock is used to generate data_clock, which carries a serialized data from the source to the destination.

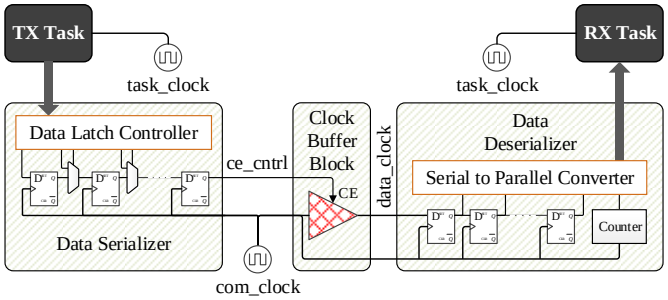

Fig. 3: Serial data transmission through clock buffers

A. Data Transfer Mechanism

The parallel data from a TX task is serialized and shifted out bit-by-bit to an RX task through the clock buffers. A register is used to latch the parallel data for onward shifting to the clock enable (CE) of the buffer on the ce_cntrl signal line. This latching is done by the Data Latch Controller. Since the same register block is used for shifting out the serial bits, multiplexers are used to select between updating the registers with new data and shifting already latched data.

The ce_cntrl signal, which carries the serial data to be transmitted controls the output of the buffer by toggling its CE. A '1' allows the input of the buffer to pass through to the output, while a '0' ties the output to zero. Since the communication clock (which can be the same as the task clock) and the task clock are synchronous, a '1' on ce_cntrl essentially allows a full clock cycle to pass through while a '0' blocks it. As an example, Fig. 4 shows the theoretically expected signal transitions for transmitting 10011010 (binary) (see Table I for the corresponding truth table). The RX task's SIPO circuit can detect a falling edge on com_clock as a '1'. With respect to the distance between the TX and RX tasks, the clock buffers in the Xilinx FPGAs are designed for short propagation delays and very low skew [7]. This helps prevent the kind of long propagation delays associated with shared-bus interconnects. As a result, the two clock signals (com_clock and data_clock) can travel far with minimal loss of phase alignment, and thus ensure timing closure.

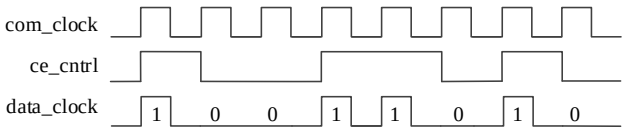

Fig. 4: An example showing the transmission of an 8-bit data 10011010

TABLE I: THE TRUTH TABLE FOR CLOCK-ENABLED DATA TRANSMISSION

Inputs		Outputs
tx_serial_data (ce_cntrl)	***com_clock***	***data_clock***
1	X	com_clock
0	X	0

B. Communication Clock and Task Clock Generation

In order to achieve the maximum possible throughput for data transfer, it is important to drive com_clock as high as possible. An advantage of using a separate clock as the communication clock is that we are not limited to the frequency of the task clock; the communication engine can run at a much higher frequency. The FPGA has PLL primitives that can be used to generate clock signals at frequencies much higher than that of the clock fed into the FPGA.

In the demonstration of CELOC in this work, the PLLE2_BASE primitive in the 7 series FPGA is used to generate the two clocks (com_clock and task_clock). Two global clock buffers are then used to distribute them throughout the chip when necessary. Fig. 5 shows the schematic of the PLL-based clock generation and distribution for CELOC. The core of the clock generator is the PLLE2_BASE primitive, which can be used as a frequency synthesizer, jitter filter, or to deskew clocks. As a clock generator, the PLL requires internal feedback as shown.

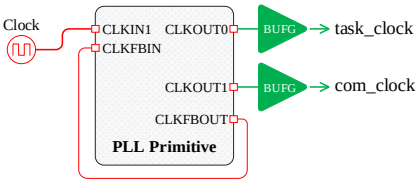

Fig. 5: The schematic of the PLL-based clock generator for CELOC

C. Clock Domain Crossing

Because com_clock and task_clock are functionally in different clock domains – one used to clock the registers that push out the serial bits, and the other to retrieve the serialized data through another set of registers, it is important to investigate the impact of *Clock Domain Crossing* (CDC). To avoid complications from CDC, the two clocks are sourced from a single PLL clock generator with the communication clock made as high as possible. This helps to prevent setup and hold timing violations by keeping both the transmission and the reception synchronous and in the same clock domain – no asynchronous clocks and no variable phase alignment. Therefore, no clock domain crossing issues are expected since the same clock (com_clock) is used to transmit and receive data [27]. Nevertheless, every implementation of CELOC is checked for CDC violations using the Vivado timing report.

D. Data Recovery Mechanism

Since we are interested in preventing static routes from crossing RM boundaries, it is important that the recovery of the serial bits at the RX input should employ a mechanism that is independent of general interconnects. Hence, an ideal interface to data_clock should be a clocking point in a logic element. Two candidates for this are registers and latches with non-clocking inputs that can be fed by clock signals. The FDPE register and the LDPE latch [28] in the 7 series FPGA fall into this category and can thus be connected as shown in Fig. 6 to receive data_clock into an RX circuit without using the general interconnect. This is because their Set/Reset (SR) and Preset inputs can be driven by global and horizontal clock buffers. Their Q outputs produce the same waveform as the original ce_cntrl signal used to toggle the clock buffers in Fig. 3. The choice of either the FDPE or the LDPE influences the maximum bandwidth of communication (see Section VII).

The FDPE is a D flip-flop with clock enable (CE) and asynchronous preset [28]. By connecting CE to a '1' and D to a '0', with the clock input fed by the same clock (com_clock) used to create the data clock at the transmitter, data_clock

connected to the PRE input produces on Q, signal level transitions corresponding to the rising edges of data_clock as shown in Fig. 7 for the same 8-bit data 10011010 (binary) transmitted in Fig. 3. To understand how this works, we consider the truth table of the FPDE (see Table II). We observe that by setting CE to '1' and D to '0', Q follows PRE (data_clock) instead of D at every rising edge of C. The LDPE data latch with asynchronous preset and gate enable [28]. A similar explanation applies to the LDPE with regards to the signal transmissions that allow data recovery, except that for the LDPE, the gate (connected to data_clock) input's signal transitions are reversed.

V. Packet Synchronization and Encoding

Since CELOC in general, serializes the data being sent before transmission, an idle line will be either a '1' or a '0'. It then becomes important for the transceiver task to determine when to start or stop reception? It is also possible for a task node to join a CELOC-based network in the middle of ongoing transmission. The new node has to correctly latch on to the beginning and end of packets. In general, a CELOC-based data transfer between any two circuits does not require any special handshaking or encoding technique, as a data bit that leaves the source circuit would arrive at the destination circuit without any ambiguity if the two circuits are directly interconnected. However, in CELOC-based NoCs, the source and destination nodes may not be directly physically connected and communication packets may be routed through intermediate nodes until they reach their destinations. As such, data packet synchronization may be required to coordinate data transfer. It may be worth noting however, that packet synchronization is not always required in bit-serial networks [8]. Depending on the adopted topology, an NoC design might be able to do away with packet synchronization and as such save on encoding resources and latency. This approach is favoured when applicable. However, in other applications, encoding is necessary to avoid ambiguity in data transfer.

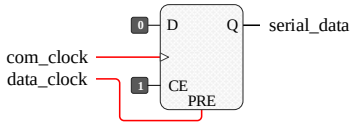

Fig. 6: The register setup for serial data recovery

Table II: The truth table of the FDPE register

Inputs				Outputs
PRE	***CE***	***D***	***C***	***Q***
1	X	X	X	1
0	0	X	X	No Change
0	1	D	1	D

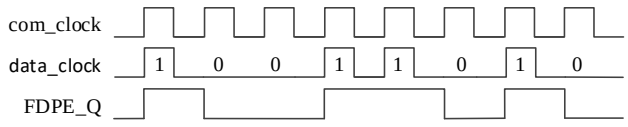

Fig. 7: A waveform showing serial data recovery from data_clock

To uniquely mark off the boundaries of transmission packets in serial networks, frame synchronization mechanisms are used. One such mechanism is byte stuffing, where a special code byte is used to delimit packet boundaries. In order to prevent incorrect synchronization, as the code byte may be present in the data packet, special 'escape' codes are often used [29], but the length of the packet ends up being inconsistent [30]. This is not desirable in real-time applications, where timeliness and predictability are important. To achieve consistency in packet size, we propose an adapted form of the *Consistent Overhead Byte Stuffing* (COBS) [30]. The COBS maps numbers in the range [0, 255] to numbers in the range [1, 255], thereby reserving one number which can be used as the frame synchronizer (delimiter). The details on how this is achieved can be found in [30]. Adopting a similar technique to map the hex number set [0, F] to [1, F], we reserve the number zero to be used as the delimiter. We call this Consistent Overhead Nibble Stuffing (CONS). Starting at the zeroth (most-significant) nibble (4 bits of "0"s and "1"s), the occurrence of a zero is replaced by the number of nibbles examined (including the zero) followed by the non-zero nibbles before the zero. For example, an arbitrary 32-bit packet of hex numbers (400AD013) passed through the CONS encoder would produce 2413AD313 (hex) as shown in Table III.

A simple way to carry out the encoding is to logically pad the packet with zero nibbles at the beginning and end as shown in the second row of Table III, with the first serving as a placeholder for the overhead and the other as a phantom helper to complete the encoding process. This phantom does not actually count as part of the data. Each nibble of the padded packet is then given an index starting at 0 from the most significant nibble (0 to 9 in this example). The encoded nibble of a zero nibble at index i_{zn} is obtained by subtracting i_{zn} from i_{znext}, where i_{znext} is the index of the next zero nibble. There cannot be another nibble after the appended zero nibble. Therefore, the encoded packet is terminated on the penultimate index (index 8 in this example). Further illustrations in Table III show that an all-zero packet would be encoded as 111111111 (hex) and a packet without a zero nibble as 9XXXXXXXX (hex), where X is a non-zero nibble.

The advantage of this form of encoding is that every packet is guaranteed to have a fixed overhead of one nibble. On the other hand, the disadvantage, as the examples show is that even when there is no zero nibble in the data, the overhead is still incurred. However, this is the price that is paid for the benefit of determinism in communication latency as far as the data packet is concerned.

Table III: Examples Showing the CONS Encoding Process

Index (i_n)	0	1	2	3	4	5	6	7	8	9
Nibbles (D_i)	0	**4**	**0**	**0**	**A**	**D**	**0**	**1**	**3**	0
Code ($i_{znext} - i_{zn}$)	2		1	3			3			
Encoded Data	**2**	**4**	**1**	**3**	**A**	**D**	**3**	**1**	**3**	
Nibbles (D_i)	0	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	0
Code ($i_{znext} - i_{zn}$)	1	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	
Encoded Data	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	
Nibbles (D_i)	0	**5**	**1**	**D**	**F**	**2**	**C**	**3**	**7**	0
Code ($i_{znext} - i_{zn}$)	9									
Encoded Data	**9**	**5**	**1**	**D**	**F**	**2**	**C**	**3**	**7**	

In order for a new node to synchronize to the communication network, a bit-level framing delimiter is required. Since there is no zero nibble in the CONS encoding, there cannot be more than three consecutive "0"s except if a

zero delimiter is used. To avoid ambiguity, a sequence of 1 and seven "0"s (10000000) will be used as the delimiter taking a cue from [30]. This delimiter or *Frame Synchronization Sequence* (FSS) is added at the beginning of each packet.

The FSS, the CONS overhead, and the data bits can all be concatenated into a single packet as shown in Table IV. The data bits can be no more than 7 bytes long for a fixed-width data packet as this is the maximum number of bytes between any two successive zeros that can be encoded by the CONS scheme. This is because the 16 nibbles in the set [0, F] are mapped to 15 nibbles in the set [1, F] for the purpose of encoding zero nibbles. In other words, only 15 consecutive zero nibbles (60 bits) can be encoded (including the phantom zero appended logically to carry out the encoding). Removing the 4 bits of this phantom zero leaves 56 bits (7 bytes) for the actual data to be transferred.

Another way of looking at this is to note from Table III that a code nibble is derived from the subtractive operation between the indexes of two nearest zeros, with the code stored in the position of the first (placeholder) zero nibble. Since a nibble can only hold a maximum count of 15 (F in hexadecimal), and the phantom zero appended to the data is part of the count, the actual data (payload), which is the allowed maximum value of the distance between any two zero nibbles, is thus 14 nibbles (index $15+i_n$ minus index i_n minus one phantom overhead nibble). This 14-nibble maximum data bits is only true for packets with infrequent zeros; if zeros are guaranteed to show up at no more than 14 nibbles apart, then a single packet can have data bits in excess of 56 bits. However, where such guarantees are not deterministic or where bounded latencies are desired as is the case with real-time networking, a fixed data bit of not more than 56 bits has to be enforced.

Nibbles have been used instead of bytes as a compromise between the percentage overhead and the maximum data bits. With a byte word length as in the original COBS, we would incur 8 bits of overhead per packet, though the maximum data bits would then be 254 bytes. A quick comparison shows that the COBS has a lower percentage overhead of 0.39% per 254 bytes compared to 7.14% per 14 bytes in CONS. However, at lower data sizes, COBS incurs more than CONS. For instance, for a data size of 6 bytes, COBS would incur 20% overhead compared to 8.33% in CONS. Moreover, the size of the delimiter also increases with the word size, always two times the word size, and thus influencing the total overhead and latency of packet transactions. Ultimately, the choice of word size will be a compromise between the percentage overhead and the maximum data bits required per packet.

TABLE IV: THE PACKET FORMAT FOR CONS-ENCODED DATA BITS

Fields	***FSS***	***CONS Overhead***	***Max. Data***
Number of Bits	8 bits	4 bits	56 bits
Comment	Value: 80 hex	CONS-encoded	

VI. NETWORK ADAPTER FOR COMMUNICATION ACCESS

To exploit the clock network for communication, each of the intercommunicating tasks in a system employing CELOC must be wrapped with a Network Adapter to arbitrate access to the CE of a clock buffer, When no packet synchronization is used, the CONS encoder and decoder are not needed and adapting to a network simply requires the *Serializer/Deserializer* (SERDES) of Fig. 3.

The popular serial communication interfaces like the Serial Peripheral Interface (SPI) and I2C are avoided because they require more than one signal. A potential interface protocol for CELOC could be a 1-wire protocol like the one introduced in [31] or the Universal Asynchronous Receiver-Transmitter (UART). Essentially, since only the CE pin in a clock buffer is being driven by an RX task in CELOC, a single-wire protocol would be more appropriate to prevent the usage of static routing resources as much as possible. The proposed SERDES provides a raw interface to the clock buffers and a higher-level bit framer can always be used to adapt to different serial protocols. As it is, the SERDES is a serial streaming interface that would bit-stream a packet of data presented at its data input and also recover a parallel data that is serially shifted in.

On the other hand, when packet synchronization is needed, the CONS encoder and decoder are used and the task is wrapped as shown in Fig. 8. This builds upon the proposed SERDES by implementing five major blocks: a CONS Decoder, a *Task Interface Logic* (TIL), and the CONS Encoder. The next subsections provide more details on these blocks and other components of the network adapter.

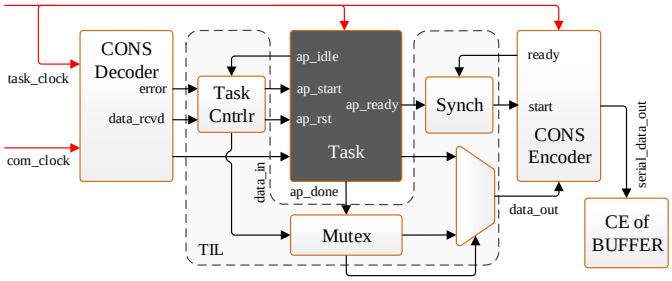

Fig. 8: The network adapter for packet-synchronized network access

A. Task Interfacing

This work proposes a task interface model that is based on the Xilinx HLS Block-Level interface protocol [32] for any task that has to communicate using CELOC's packet-synchronized wrapper. This model requires that a minimum of five ports: ap_idle, ap_start, ap_rst, ap_ready, ap_done, and ap_return (indicated by the arrow that feeds the Mux in Fig. 8) are defined for a task. This ensures uniformity of interfacing between different tasks and the CONS codec (decoder-encoder) and provides a standardized task interface. In addition, this is also in line with the current trend in using HLS-generated HDL modules for rapid system development.

B. CONS Encoding

In the encoder, which also serves the function of data serializer, the CONS encoding algorithm is implemented with a finite state machine. The theoretical encoding process presented in Section V is modified for hardware implementation. The encoder starts its operation when a START signal is asserted. First, it saves the data to encode in a shift register and then starts the encoding process. The process involves detecting zero nibbles and replacing them with CONS codes. Once the entire packet is encoded, the bits are shifted out serially for routing to the CE of a clock buffer.

When all the bits have been shifted out, the encoder asserts a READY signal in readiness for another encoding operation.

C. CONS Decoding

The CONS decoder receives, decodes, and de-serializes a CONS-encoded packet. In the decoder, the code nibbles in the received packet are replaced with zeros. The decoding is simplified by careful implementation of the decoding algorithm. A close look at the encoded data in Table III reveals that every code nibble points to the relative location of the next code nibble. This is as expected since the CONS codes are formed by counting the number of non-zero nibbles preceding a zero nibble as explained in Section V. Also, the first nibble received is always a code nibble. These observations are crucial as they simplify the logic of the decoder, and hence reduce the FPGA resources used. By subtracting 1 from the value of a code we obtain the number of data nibbles preceding the next code. Using a state machine, we loop through all the codes and extract the associated data. Once all the nibbles have been processed the data_rcvd port (see Fig. 8) of the decoder is asserted and the state machine resets in anticipation of a new packet.

D. Address-Inclusive Encoding and Decoding

A generic approach is taken in the implementation of the encoder so that it is usable for varying numbers of data width. In the vanilla implementation, where there is no addressing or any special bits inserted in the packet, the codec is intended for P2P communication without provision for addressing. However, the applications that require CELOC for use in an NoC would benefit from an addressable codec that has the addressing functionality embedded in it.

More often than not, the packet in Table IV will need addressing if CELOC is used to implement an on-chip network. In appending the address bits to the packet, a plain un-encoded non-zero addressing is recommended, where a zero address is not used and the address bits, therefore, do not need to be encoded. This ensures the address bits do not eat into the maximum number of bits available for data. More important though, is the fact that in a CELOC-based NoC, the packet can, therefore, be routed through the network with much less latency since the intermediate nodes do not necessarily have to receive the entire packet as the address to deflect a packet to is visible in the packet. There is a use_addr port, with corresponding address ports on the interface of the encoder and the decoder. This is used to enable the address-inclusive mode and is controlled by the TIL.

E. Task Interface Logic

The TIL interfaces the task to the CONS Encoder and Decoder. It glues together a *Task Controller* (TC), Mutex, multiplexer (Mux), and Synchronizer (Synch). The Mutex is a means of sending status information out of the task and wrapper, especially for the purpose of error detection. For instance, the decoder could fail due to an error in its internal state machine's state transition. The Mux is used to choose between the outputs of the task and the Mutex.

The function of the TC is to start the task if it is idle and data has been received by the decoder. It also deserializes the received packet, recovers the data and presents it to the task. In addition, it handles addressing when the address-inclusive CONS is used. These are the functions of the TC in this prototypic implementation. However, in a reconfigurable system that deploys CELOC, the TC would receive command packets from a system-level *Task Communication Manager* (TCM) to start or stop its associated task. It would also receive the destination address and the system time instance the processed data should be sent. This time, however, cannot be earlier than the time instance the task finishes execution. If the packet is a command packet, the TC would check whether to start the task or reset it based on the command and would do accordingly. Otherwise, the received packet would be handled as a data packet. If the task is already started, the TC would route the new data to the task. The situation should not arise where a task is not ready for new data, thus avoiding the need for buffering and saving on memory resources. This is because the TCM would dictate the time to send data based on when a destination task can accept it. It would, therefore, be counterintuitive to provide a buffering capability. However, a buffer can easily be inserted if necessary but the TCM's algorithm and the task computation model would have to be modified to account for this. In general, data should not be processed by the task at a rate faster than it can be routed through the CONS Encoder/Decoder (Codec) and the serial communication network except if buffering is used.

Similarly, in data delivery to the CONS Encoder is not buffered. The TIL ensures that the encoder is ready for a new input before applying the task's output data. The Synchronizer does this by checking that both ap_ready and CONS Encoder's "ready" are driven HIGH before asserting the CONS Encoder's start. It is guaranteed that once ap_ready goes HIGH, the data from the task is available as input to the CONS encoder. This is because the Output Data Mutex and Multiplexer are purely combinatorial and as such incur no clock delays. To ensure a non-buffered data at the input of the encoder, the encoding time should be accounted for in the timing model a system deploying CELOC.

F. Resource Utilization and Performance Evaluation

Table V shows the resource overhead of the network adapter for the bare SERDES and the packet-synchronized version. Tiny finite state machines are implemented for the PISO and SIPO blocks of the SERDES. These incur a total of 13 slices while the CONS-based adapter's utilization amounts to only 32 slices. As the reconfiguration frame, which defines the minimum selectable area for partitioning the FPGA area, is always two columns (200 slices for CLBs). This implies that the adapter will fit right in, even with the smallest of tasks, by occupying only between 6.5% and 16% area of the reconfiguration frame.

TABLE V: RESOURCE UTILIZATION OF CELOC'S NETWORK ADAPTER

Resource	Network Adapter Modules				
	SERDES		Packet-Synchronized		
	PISO	*SIPO*	*CONS ENC*	*CONS DEC*	*CONS TIL*
FFs	30	72	65	100	84
LUTs	14	10	35	28	24
DSPs	0	0	0	0	0
BRAMs	0	0	0	0	0
Slice	13		32		

In terms of data transfer latency, the SERDES PISO block latches the parallel data in one clock cycle and streams it for the number of clock cycles equivalent to the number of bits in the parallel data. The SIPO recovers the data in the same period but uses one additional clock cycle for internal state transitions. As a result, the SERDES latency for 32-bit data is 34 clock cycles.

For the evaluation of communication latency of the packet-synchronized adapter, the CONS encoder is directly interfaced to the CONS decoder and the number of cycles measured individually for the encoder and decoder; and also for the entire codec. Table VI presents the measured clock cycles. From the moment the encoder's start signal is asserted to the end of encoding and serial data shifting out (ready signal asserted), 56 clock cycles are incurred for the non-addressable codec and 60 for the address-inclusive version. Similarly, for the decoder, from the moment the FSS is received to the end of decoding, these numbers are 48 and 52 respectively. An entire 32-bit packet is sent and received in 60 and 64 clock cycles respectively.

For the different word sizes, the test packet is made a multiple of 32 bits for ease of comparison and the generation of the latency equations. That is, for the word lengths of 4, 8, 16, and 32, data widths of 32, 64, 128, and 256 bits are used. The equations should be used only when the packet size is $8N_W$, where N_W is the word length in bits.

TABLE VI: THE CELOC CODEC'S LATENCIES FOR DIFFERENT WORD SIZES

Word Size (N_W) in Bits	**Max. Data Size Per Packet**	**Clock Cycle Latency for $8N_W$ Data Bits**					
		Non-Addressable			**4-bit Addressable**		
		ENC	***DEC***	***CODEC***	***ENC***	***DEC***	***CODEC***
4	56 bits	56	48	60	60	52	64
8	254 bytes	100	88	108	104	92	112
16	128 kB	188	168	204	192	172	208
32	16,384 MB	364	328	396	368	332	400
Latency Equation		$60 + 12(N_W - 4)$			$64 + 12(N_W - 4)$		

VII. BUFFER CONFIGURATIONS FOR NETWORK ACCESS

The diagram in Fig. 9 is a representation of a section of the Xilinx 7 series FPGA. The left and right clock regions are symmetrical with respect to the placement of clock buffers. Also indicated, are *Circuit Regions* (CRs) that are potential areas within the clock regions for task placement targeting the exploitation of clock buffers for communication. Note that not all possible CRs and communication routes are shown. The notion here is that inter-circuit communication should only be through the clock buffers for the purpose of avoiding the static routes completely in order to aid the flexible relocation of circuits. Fig. 9 also shows the locations of the clock buffers on the FPGA. In general, a 7 series FPGA (apart from the smaller Artix-7 chips) has 4 BUFRs, 2 BUFMRs, and 12 BUFHs in each clock region. In addition, there are 32 BUFGs at the centre of the chip, common to all the regions.

While the buffers can be used to communicate in different directions, certain configurations or combinations have to be used in order to provide communication. This brings the question of data transfer speed since by connecting one buffer to another, a delay is introduced into the communication path. In the next subsections, we present some possible configurations of the clock buffers and their use cases. What informs these configurations are the reach of the buffers and the availability on them of clock enable (CE) pins that can be actively toggled. Furthermore, there is a restriction on buffer-to-buffer interconnections. For instance, it is only a BUFR that can feed a BUFG directly. As a result, if an intra-clock-region network has to access other clock regions not within its neighbourhood (immediate vertical and horizontal regions), a BUFR has to be used to drive a BUFG in order to feed such regions with data_clock.

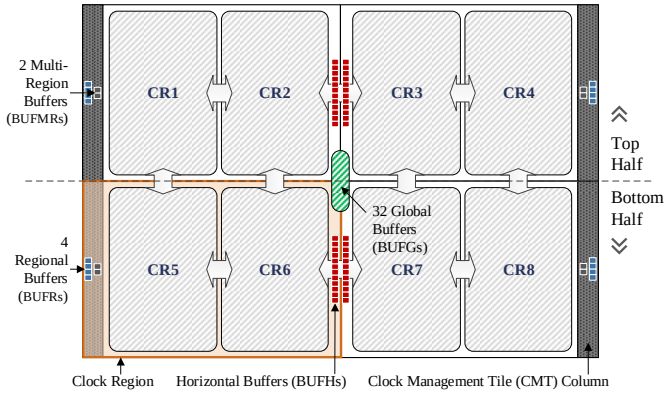

Fig. 9: A representation of the 7 series FPGA chip showing the locations of clock buffers, circuit regions (CR1 to CR8) for placing circuits within clock regions, and sample vertical and horizontal interconnections between the CRs

Since only the BUFGs and BUFHs can drive the SR/PRE and CE inputs of registers, and it is essential for data_clock to be interfaced with a receiving circuit through an input that can be driven by a clock buffer, not all of the possible buffer combinations can be used if relocation support is sought. Only the ones that have data_clock coming out of a BUFG or BUFH can be used.

In the subsections that follow, it should be noted that arrows represent the direction of data transfer. For instance, with respect to Fig. 9, CR1→CR8 means CR1 is transmitting to CR8 while CR2↔CR3 implies a bidirectional data transfer between CR2 and CR3. Moreover, since the clock regions have some symmetry, a communication like CR1↔CR3 is treated the same as CR2↔CR4; and CR1→CR6 as CR5→CR2. Also, except where indicated in the figures, all the connected clock buffers are in the same clock region, save for the BUFGs, which do not belong to any clock region.

A. Clock Buffer Configurations for Global Communication

Since the BUFGs have a device-wide reach, they can be used to transmit data to anywhere on the chip. This prevents any circuit region from being excommunicated from other regions. For instance, to communicate directly with CR-8 from CR-1 in Fig. 9, a BUFG has to be used (see Fig. 10(a)). However, since the BUFGs are at the centre of the chip, an intermediate buffer has to be used to get access to them. There are four options – BUFR, BUFMR→BUFR (used in Fig. 10(a)), BUFH→BUFR, or BUFH→BUFMR→BUFR, depending on the available buffers and the relative location of the task. In contrast, for global communication like CR2↔CR7, there is direct access to BUFGs. Therefore, the BUFG-only configuration shown in Fig. 10(b) can be used. Note that any communication that does not reside within the

same clock region, or that reaches only to an immediate vertical or horizontal region is classified as global.

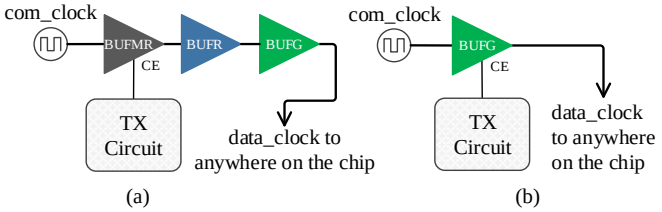

Fig. 10: Clock buffer configurations for global communication showing (a) [BUFMR→BUFR→BUFG→], and (b) [BUFG→]

B. *Buffer Configurations for Horizontal Communication*

By placing circuits at the inner edges of two horizontal adjacent clock regions, advantage can be taken of the BUFHs for horizontal communication. However, circuits at the outer edges require other configurations. With respect to Fig. 9, and picking the top clock regions for illustration, the communications that fall into the category of the horizontal inter-clock region include CR1→CR2, CR1→CR3, CR1→CR4, CR2→CR3, and CR2→CR4. Communications like CR2↔CR3 and CR2→CR1 can be achieved by using the configuration in Fig. 11(a). However, since there is no clock-buffer-based physical link between CR1 and CR2 in the middle of the clock region, while CR2→CR1 can use [BUFH→], CR1→CR2 can use [BUFMR→BUFR→] (see Fig. 11(b)). All the other communications have to use the configuration [BUFMR→BUFR→BUFG→] presented in Fig. 10(a). It should be noted that CR2→CR1 can also use the configurations in Fig. 11(c) and Fig. 11(d).

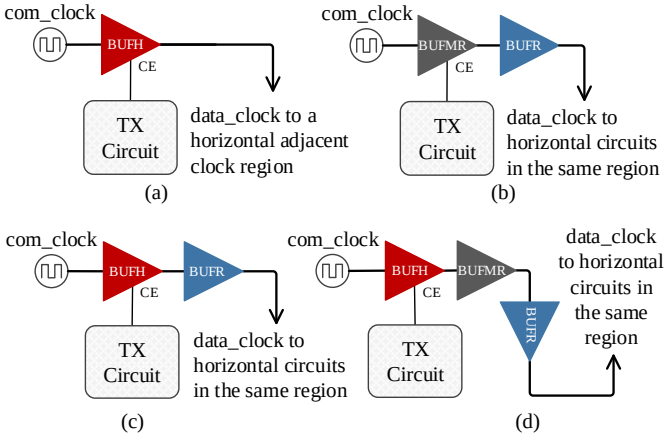

Fig. 11: Clock buffer configurations for horizontal inter-region communication showing (a) [BUFH→], (b) [BUFMR→BUFR→], (c) [BUFH→ BUFR→], and (d) [BUFH→BUFMR→BUFR→]

C. *Clock Buffer Configurations for Vertical Communication*

From Fig. 9, and picking the left clock regions for illustration, the communications that fall into the category of the vertical inter-clock region include CR1↔CR5, CR1→CR6, CR2→CR5, and CR2↔CR6. Note that symmetry can be used to pick other ones. CR2→CR5, CR6→CR1 and CR2↔CR6 communications can be through the configuration in Fig. 12(a) and [BUFG→] presented in Fig. 10(b). All others can use the one in Fig. 12(b). Note that the BUFRs in Fig. 12 are not in the same clock region as the other buffers. As a result, they are indicated as BUFR(adj) in the figure caption.

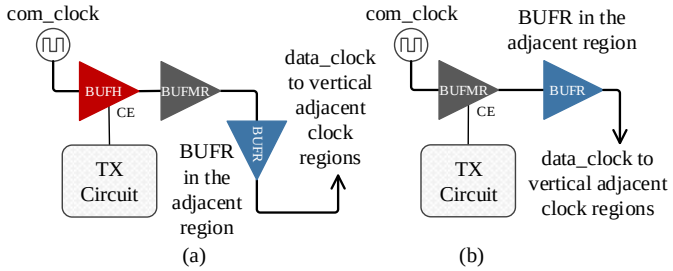

Fig. 12: Clock buffer configurations for vertical communication showing (a) [BUFH→BUFMR→BUFR(adj)→], and (b) [BUFMR→ BUFR(adj)→]

D. *Maximum Speeds of the Clock Buffers and Nets*

The understanding of the maximum frequency of operation of the clock buffers is crucial in the attempt to adapt them for on-chip communication. This will help evaluate the limiting factor in the achievable communication speed of CELOC for different devices. Table VII shows the maximum frequencies found in the respective datasheets of the 7-Series FPGAs. At the time of writing, there is no Spartan-7 with speed grade -3.

E. *Bandwidth Characterization*

To determine the maximum speed that each clock buffer configuration can achieve, the experimental setup in Fig. 13 is used. The device used for the experiment is the Artix-7 (xc7a35tcpg236-1) FPGA with speed grade -1. The TX circuit is used to generate predetermined data packets to be received by the RX circuit. To confirm the correctness of data transfer, the validation of the received data is done by using an *Integrated Logic Analyser* (ILA) to observe the signal transitions. A PLL clock generator is used to sweep the communication clock's frequency to the maximum value that still meets timing and does not corrupt the communication.

TABLE VII: MAXIMUM OPERATING FREQUENCIES OF THE CLOCK BUFFERS IN THE XILINX 7 SERIES FPGAS

Device	Speed Grade	BUFG [Tree] (*MHz*)	BUFH [Buffer] (*MHz*)	BUFMR [Buffer] (*MHz*)	BUFR [Tree] (*MHz*)
Artix-7	-3	628	628	680	420
Kintex-7		741	741	800	600
Virtex-7		741	741	800	600
Spartan-7	-2	628	628	-	375
Artix-7		628 (394[a])	628 (394[a])	680 (600[a])	375 (315[a])
Kintex-7		710 (560[a])	710 (560[a])	800 (667[a])	540 (450[a])
Virtex-7		710	710	800	540
Spartan-7	-1	464	464	-	315
Artix-7		464	464	600	315
Kintex-7		625	625	710	450
Virtex-7		625	625	710	450

[a] At speed grade -2LE, *0.9 V*

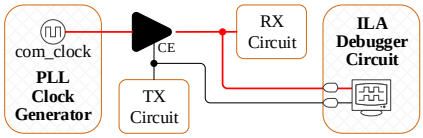

Fig. 13: The setup for characterizing the clock buffer configurations

The test communication packet is a 44-bit packet comprising of 32-bit data, a frame synchronization sequence of 8 bits and a serial encoder overhead of 4 bits. Table VIII

shows the maximum bandwidths (data transfer rates) at which the buffer configurations work without a corrupted data transfer. The corresponding data recovery register or latch is also indicated. Moreover, based on the number of clock buffers in the FPGA, the number of instances of each configuration (not considering buffers used by other configurations and those used for other purposes) that can coexist in a single clock region is also indicated in Table VIII.

The highest speed of 275 Mbps is observed with the [BUFH→BUFR→] configuration and the lowest (171.43 Mbps) with the [BUFR → BUFG →] and [BUFMR → BUFR → BUFG→] configurations. Indeed, BUFR appears to be the limiting buffer. This is because it is the buffer with the slowest speed. In fact, for the Artix-7 board used for the experimentation, the speed grade is -1 and the BUFR's maximum frequency is graded at 315 MHz, which is the minimum for any of the buffers in the chip (see Table VII). Therefore, the average speed of 221.15 Mbps is relatively high considering that it is only 29.79% short of the BUFR's maximum. By extension to other speed grades, the average CELOC speed can be expected to scale as 70.21% of f_{BUFR_MAX}, where f_{BUFR_MAX} is the maximum frequency of the BUFR net in the target device.

TABLE VIII: BANDWIDTH OF THE CLOCK BUFFER CONFIGURATIONS

Clock Buffer Configuration	FDPE /LDPE	Bandwidth (Mbps)	Instance/ Region
BUFG→	FDPE	266.67	32/chip
BUFR→BUFG→	LDPE	171.43	4
BUFMR→BUFR→BUFG→	LDPE	171.43	2
BUFH→BUFR→BUFG→	LDCE	187.50	4
BUFH→BUFMR→BUFR→BUFG→	LDCE	171.43	2
BUFH→BUFMR→BUFR(adj)→BUFG→	LDCE	171.43	2
BUFH→	FDPE	266.67	12
BUFMR→BUFR→	LDPE	240.00	2
BUFH→BUFR→	LDPE	275.00	4
BUFH→BUFMR→BUFR→	LDPE	240.00	2
BUFH→BUFMR→BUFR(adj)→	LDPE	240.00	2
BUFMR→BUFR(adj)→	LDPE	240.00	2
BUFR→	LDPE	233.33	4
Average	-	**221.15**	-

In addition, the use of the clock buffers does not impact negatively on the number of circuits that can be on the FPGA simultaneously. The CRs can accommodate more than one circuit. Therefore, from the number of instances in Table VIII, it is estimated that up to 16 circuits with communication access can be in a single clock region at the same time assuming they can be partitioned such that the CE access does not constitute a static route problem. This number is arrived at by excluding configuration instances that use similar buffers. In addition, there are 32 global configuration instances that can be shared by all the clock regions.

VIII. DYNAMIC COMMUNICATION WITH CELOC

This section advances a dynamic communication access mechanism that is termed *Clock-Enabled Relocation-Aware Network-on-Chip (CERANoC)*. This builds on CELOC with the main advantage for CERANoC being that the clock buffers and nets use dedicated routes that are independent of the general logic interconnect. This removes the restriction of the static interconnect links and enhances the online relocation of circuits. This mechanism relies on the replacement of the interconnect links in NoCs with clock buffers. Since the clock buffers do not use the general logic routing resources, the path from a transmitting circuit to a receiving circuit is free of general logic interconnections.

For circuit relocation to be feasible, communication must be provided at the resource-matching destination for the circuit being moved. With regard to Fig. 14, the easiest way to provide this communication is to ensure that a route from Task 1 to LOC 2 is established at design time. This way, during runtime, Task 2 can be moved to LOC 2 while maintaining its communication link with Task 1.

The solution to dynamic communication in CERANoC eliminates the static inter-circuit communication routes altogether. Using Fig. 14 as an example, this is achieved by removing the static inter-task connections and replacing them with clock buffers as shown in Fig. 15. The hypothetical layout of tasks here is the same as that in Fig. 14, except that the interfaces between Tasks 1 & 2, and between Tasks 1 & 3 have been removed. To provide communication, a clock buffer is used to transmit serial bits from Tasks 1 to 2 and 3. This signal also feeds LOC 2, so that if Task 2 is relocated to LOC 2, the communication between it and Task 1 remains intact. At the same time, LOC 1 is now free of a crossing routing. Basically, the surface of the chip is generally freed of inter-circuit routings.

In the implementation of CERANoC in this work, the clock regions of the FPGA are used as NoC nodes. The clock buffers are pre-routed between clock regions at design time so that during runtime, regardless of the clock region a task is placed, it is able to communicate with any task in any other clock region, with serial data riding on a clock signal from a source node to a destination node.

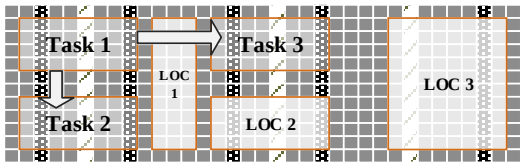

Fig. 14: A diagram demonstrating how static routes hinder relocation

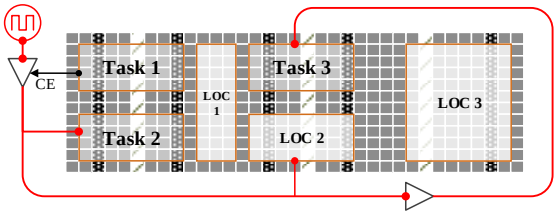

Fig. 15: By removing the inter-circuit interfaces and replacing them with clock buffers, it is possible to achieve dynamic communication

In the following subsections, the special considerations for the implementation of CERANoC are presented in relation to the design parameters of a traditional NoC. It should be noted that the aim is not to implement a full-fledge NoC as that is beyond the scope of this work; rather, the intention is to demonstrate how the use of clock buffers for inter-circuit communication enhances the relocatability of circuits.

A. Packet Format and Addressing Scheme

The FSS, the destination address, the CONS overhead, and a data of 56 bits maximum are all concatenated into a single packet as shown in Table IX. A unique Node ID is given to each node on the network. This ID also serves as the address of the node and is added to the communication packet as the destination address. With *N* nodes, the address range is from 1 to *N*, with zero deliberately avoided because the address field in the packet is un-encoded as noted for the address-inclusive packet encoding and decoding in Section VI. As such, one reason for transmitting the address in plain format is that the routers need to know the destination address before routing the packet. Encoding and decoding the address would incur further clock cycles at the encoder and decoder and require more logic resources in the router. Furthermore, an encoded address would eat into the number of bits available for the actual data, eventually preventing data transfer as *N* increases and becomes 56. Therefore, with this careful choice of an address range, CERANoC saves on time and resources, and ensures maximum throughput.

TABLE IX: THE PACKET FORMAT FOR 4-BIT-DATA-WORD CERANOC

Fields	***FSS***	***Destination Address***	***CONS Overhead***	***Data***
Number of Bits	8 bits	[a]*N* bits	4 bits	56 bits max
Description	Value: 80 hex	Unencoded	CONS-encoded	

[a] *N* = number of nodes

B. Network Routing

A routing algorithm determines the routing of data from the source to the destination in a network. The problem of designing routing algorithms that meet different performance and architectural requirements has been extensively studied. Some of these requirements are low latency, low power consumption, scalability, and programmability [33]. CERANoC supports any existing routing algorithm so far the clock buffers can be arranged to serve as links in the topology chosen. There is no other special consideration for routing in CERANoC. For instance, a Torus CERANoC can use BUFGs to connect the topmost clock regions to the tail regions and the leftmost regions to the rightmost regions.

C. Prototype Network Demonstration

To demonstrate the feasibility of CERANoC, a 4-node prototype star network with a Central Router is implemented on the Artix-7 (XC7A35TCPG236) FPGA chip (see Fig. 16). For the global clock generation and distribution, a special switch_clock is needed by the Central Router. This is from the same output that feeds the BUFG which distributes com_clock. The clock buffer configuration used by the nodes is [BUFMR→BUFR→BUFG→] but could have also been [BUFR→BUFG→]. Note how the BUFGs are located logically inside the Central Router. Because switch_clock feeds these BUFGs, passing it through another BUFG would adversely affect network bandwidth. Moreover, by directly feeding the BUFGs, switch_clock ensures that on the part of a receiving node, a received packet arrives directly from another node with the data clock having been refreshed. This is because as the data_clock from the BUFR enters the Switch Arbiter, it is received by an LDPE latch and passed through a crossbar logic before being fed to the CE of a BUFG, essentially starting a new transmission (see Fig. 17).

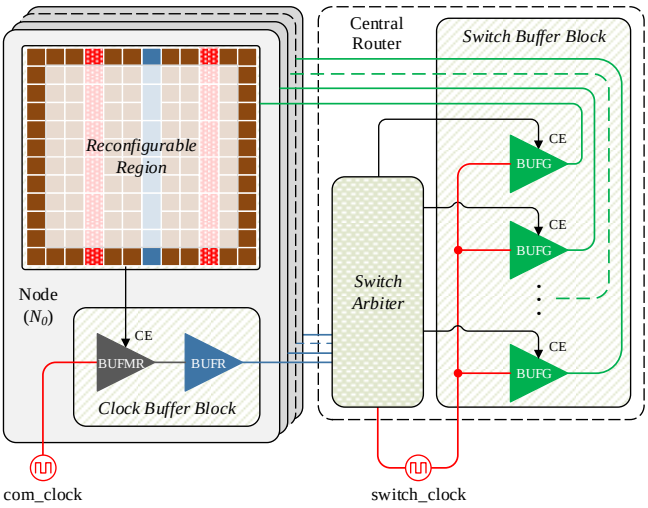

Fig. 16: A 4-node CERANoC star network using clock buffers as links

Since the objective of this prototype demonstration is to show that using the clock buffers in the manner stipulated by CELOC/CERANoC facilitates dynamic communication and circuit relocation by circumventing the general interconnect, most of the intricacies of NoC designs are avoided as these are already extensively studied. A point-to-point routing is used for the star-network (see Fig. 18), and a 32-bit payload data is used, giving a 48-bit packet. 48-bit buffer memories (48 x 1-bit LUT-RAMs) are provided inside the Central Router in order to temporarily store packets that cannot be immediately routed. In order to control access to multiple nodes attempting to transfer packets simultaneously to the same receiving node and thus keep in line with real-time requirements, priorities are assigned to the nodes based on the node address. A node with a lower address has a higher priority.

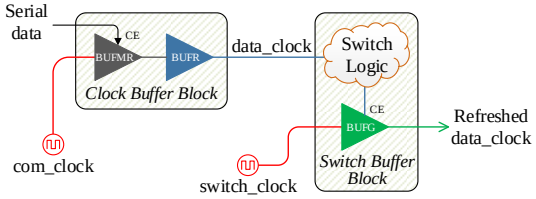

Fig. 17: Data clock renewal in a star-shaped CERANoC

Fig. 18 shows the switch architecture implemented for the 4-node star network. N_i implies node i while S_{jk} denotes a switch position from node j to node k. The indicated positions of the switches are for the following routing: $N_0 \rightarrow N_3$, $N_1 \rightarrow N_0$, and $N_3 \rightarrow N_1$. There is one *Node Router* (NR) for each input to the Switch Arbiter. In each NR there are three $(N-1)$ independent switch endpoints (S_{jk}) which determine the routing of the incoming packet to the other three nodes. In the Switch Arbiter, there is a 4-bit *occupied_switches* register that shows the state of the nodes with respect to data reception. A node that is presently receiving a packet has its corresponding bit turned on. The Switch Arbiter checks the destination address of the packet against the state of the occupied switches. If the destination node is not already occupied by an

ongoing transmission, the packet is routed through and the *occupied_switches* state is updated.

To test dynamic communication and relocation at the fundamental level, four tasks (θ_0 to θ_3) are set up, with one task in each of the nodes. It is very important in this demonstration to have a visual indication that new tasks are able to establish communication and that existing tasks still execute correctly when new tasks are placed in runtime. As a result, a VGA application is used with the setup in Fig. 19.

Tasks θ_0 to θ_2 are pattern generators, each generating three different patterns (P) of four vertical stripes of colours white (W), red (R), green (G), or blue (B). Each of these coloured stripes is represented by 8 bits (3 bits for R, 3 bits for G, and 2 bits for B). Every 32 bits of data sent by a pattern generator, therefore, determines four stripes of 8-bit colour. θ_0 to θ_2 generate P_0 to P_2 respectively, with P_0 = [W, R, G, B], P_1 = [G, B, W, R], and P_2 = [B, W, R, G]. θ_3 is a fixed VGA controller that interfaces to a VGA monitor in order to display the patterns generated by θ_0 to θ_2. At design time θ_0 to θ_3 are floor-planned in nodes N_0 to N_3 respectively and partial bitstreams are generated for only θ_0 to θ_2. Task θ_3 has to be static because it needs access to the VGA's interface pins which are in fixed locations on the FPGA. Tasks θ_0 to θ_2 are set to transmit to θ_3 at the same time. Because of the router priority, this means P_0 is continuously displayed. By blanking N_0 and N_1 successively using blanking bitstreams, we are able to see P_1 and P_2; reconfiguring N_1 then N_0 also results in patterns P_1 then P_0, demonstrating that communication is unimpaired when tasks are swapped in and out in runtime.

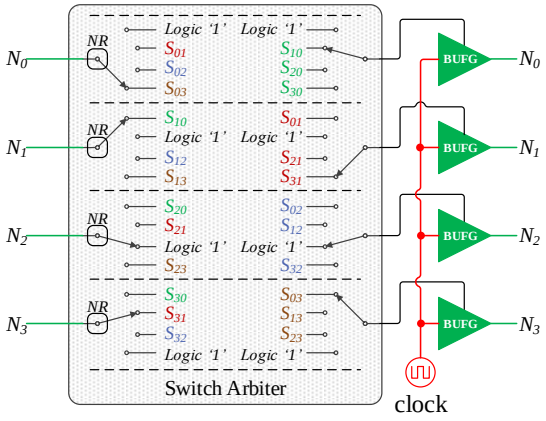

Fig. 18: The switch architecture for a 4-node CERANoC star network

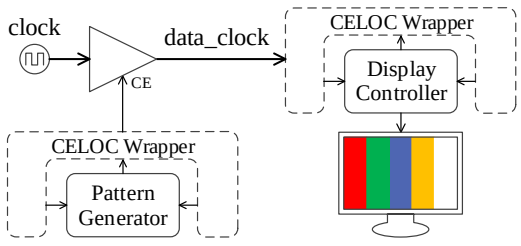

Fig. 19: The setup for demonstrating CERANoC

The demonstration of relocation involves configuring θ_0 in N_1 while blanking N_0 and θ_1 in N_0 while blanking N_1, though, after changing the target frame address in the bitstream. In the former case, we are able to see pattern P_0 even though it is configured in N_1, and vice-versa for the latter case. The *Vivado Hardware Manager* is used to configure the partial bitstreams. Fig. 20 shows the floorplan of the FPGA after implementation. It can be seen that the chip areas belonging to nodes 0 to 2 are free of general routing. This is as expected. The Network Interface does not contribute to static routing as it is made part of the reconfigurable task itself. Only the clock lines can be seen routed in the HROW from a global network feeding the clock regions (refer to Fig. 2 in Section III). These routings are dedicated clock nets and do not interfere with relocation. This means the clock regions remain free of general routing even though they are interconnected. The connections to CEs are at the edges of the clock regions, leaving the majority of the region free of general routing.

D. Resource Utilization

The only component peculiar to CERANoC is the network adapter and it uses only 32 slices (see Section VI). The entire 4-node network itself (without the tasks) takes 144 slices. Clock buffer utilization stands at 4 BUFMRs, 4 BUFRs, and 6 BUFGs, with per clock region utilization of 50%, 25%, and 3.125% respectively.

E. Network Latency

Since the central router simply routes the packet from the source to the destination nodes, essentially effecting the connection between [BUFMR→ BUFR→] and [BUFG→] in a [BUFMR→BUFR→BUFG→] clock buffer configuration, the packet transfer latency remains 64 clock cycles as presented in Table VI in Section VI for packet-synchronized address-inclusive encoding.

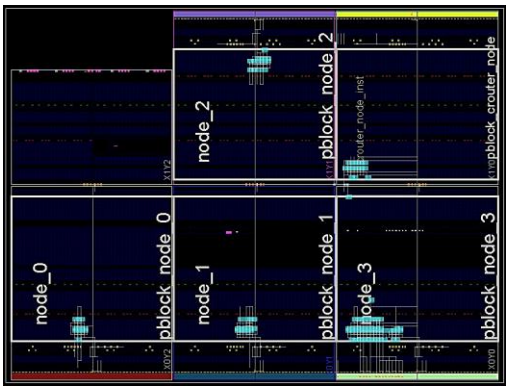

Fig. 20: The floorplan of the implemented 4-node network

F. Network Throughput

The CONS encoder and decoder do not share circuitry. Therefore, nothing stops concurrent data transfers like these four simultaneous data transfers: $N_0 \rightarrow N_1$, $N_1 \rightarrow N_2$, $N_2 \rightarrow N_3$, and $N_3 \rightarrow N_0$. That is, for the 4-node star CERANoC, the throughput of the individual link can be multiplied by 4 to obtain the network throughput. As such, for an N-node star CERANoC in full-duplex mode, the throughput (in Mbps) can be defined by Eq. 1 in terms of the payload size (in bits), the number of nodes (N), the frequency of operation (f in MHz), and the latency cycles as follows:

$$Throughput = \frac{Payload\ (in\ bits) \times N \times f\ (in\ MHz)}{Latency\ Cycles} \qquad (1)$$

At 100 MHz this gives a throughput (data rate) of 200 Mbps for the network demonstrated. The CELOC links used has a maximum speed of 171.43 Mbps (same as 171.43 MHz since one bit is transmitted in one clock cycle). The maximum throughput for the Artix-7 device used is, therefore, 428.58 Mbps for a 32-bit payload and $N = 5$ (assuming the *Central Router*'s RP is also used to host a node). It should be noted that the latencies for payloads other than 32 bits can easily be determined from Table VI.

Compared with methods that involve runtime routing, CERANoC does not incur any clock cycle overhead in order to place a new circuit or relocate one in runtime. Moreover, compared with the method in [22], the ICAP is not required for communication purposes, thus allowing SEM to have as much ICAP time as possible. The use of the ICAP for communication could be counterintuitive where reliability is important. Moreover, while DyNoC [19] also achieves dynamic communication for newly placed tasks, it is not certain that it is able to support relocation since the problem of general routing seemed not to have been addressed. CERANoC, on the other hand, leaves the chip area clear of general routing.

IX. Summary and Future Work

This article has presented a unique adaptation of the clock buffers and nets of an FPGA for dynamic communication for relocatable circuits. By using clock buffers as communication infrastructures, we have shown that it is possible to avoid static interconnections and achieve communication among existing tasks and tasks placed or relocated during run time on an FPGA. One limitation in the present implementation arises from the limited number of clock buffers, limiting the network throughput. This is alleviated in newer chips like the UltraScale, which have more clock buffers. In the future, we will investigate architectures that will further exploit the clock buffers for dynamic communication.

References

[1] P. Sedcole, B. Blodget, J. Anderson, P. Lysaghi, and T. Becker, 'Modular partial reconfigurable in Virtex FPGAs', in *International Conference on Field Programmable Logic and Applications, 2005.*, 2005, pp. 211–216.

[2] S. Srinivasan *et al.*, 'Toward Increasing FPGA Lifetime', *IEEE Trans. Dependable Secure Comput.*, vol. 5, no. 2, pp. 115–127, Apr. 2008.

[3] L. Kirischian, V. Kirischian, and D. Sharma, 'Mitigation of Thermo-cycling effects in Flip-chip FPGA-based Space-borne Systems by Cyclic On-chip Task Relocation', in *2018 NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, 2018, pp. 17–24.

[4] Xilinx Inc., 'Vivado Design Suite User Guide, Partial Reconfiguration - UG909 (v2018.1)'. Xilinx Inc., 2018.

[5] A. DeHon, R. Huang, and J. Wawrzynek, 'Hardware-assisted fast routing', in *Proceedings. 10th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, 2002, pp. 205–215.

[6] W. J. Dally and B. Towles, 'Route packets, not wires: on-chip interconnection networks', in *Design Automation Conference, 2001. Proceedings*, 2001, pp. 684–689.

[7] Xilinx Inc., '7 Series FPGAs Clocking Resources - User Guide UG472 (v1.11.2)'. Xilinx Inc., 2015.

[8] N. Kapre, 'On Bit-Serial NoCs for FPGAs', in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2017, pp. 32–39.

[9] A. Morgenshtein, I. Cidon, A. Kolodny, and R. Ginosar, 'Comparative analysis of serial vs parallel links in NoC', in *2004 International Symposium on System-on-Chip*, 2004, pp. 185–188.

[10] A. Adetomi, G. Enemali, and T. Arslan, 'Clock Buffers, Nets, and Trees for On-Chip Communication: A Novel Network Access Technique in FPGAs', in *2017 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, 2017, pp. 219–222.

[11] A. Adetomi, G. Enemali, and T. Arslan, 'Relocation-Aware Communication Network for Circuits on Xilinx FPGAs', in *2017 27th International Conference on Field Programmable Logic and Applications (FPL)*, 2017, pp. 1–7.

[12] A. Adetomi, G. Enemali, and T. Arslan, 'Characterization of Clock Buffers for On-Chip Inter-Circuit Communication in Xilinx FPGAs', in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2018, pp. 1–5.

[13] T. S. T. Mak, P. Sedcole, P. Y. K. Cheung, and W. Luk, 'On-FPGA Communication Architectures and Design Factors', in *2006 International Conference on Field Programmable Logic and Applications*, 2006, pp. 1–8.

[14] B. Fu and P. Ampadu, 'Networks-on-Chip (NoC)', in *Error Control for Network-on-Chip Links*, Springer New York, 2012, pp. 33–47.

[15] T. Bjerregaard and S. Mahadevan, 'A Survey of Research and Practices of Network-on-chip', *ACM Comput Surv*, vol. 38, no. 1, Jun. 2006.

[16] É. Cota, A. de M. Amory, and M. S. Lubaszewski, 'NoC Basics', in *Reliability, Availability and Serviceability of Networks-on-Chip*, Springer US, 2012, pp. 11–24.

[17] T. Marescaux, A. Bartic, D. Verkest, S. Vernalde, and R. Lauwereins, 'Interconnection Networks Enable Fine-Grain Dynamic Multi-tasking on FPGAs', in *Field-Programmable Logic and Applications: Reconfigurable Computing Is Going Mainstream*, M. Glesner, P. Zipf, and M. Renovell, Eds. Springer Berlin Heidelberg, 2002, pp. 795–805.

[18] F. Alazemi, A. AziziMazreah, B. Bose, and L. Chen, 'Routerless Network-on-Chip', in *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 492–503.

[19] C. Bobda, M. Majer, D. Koch, A. Ahmadinia, and J. Teich, 'A Dynamic NoC Approach for Communication in Reconfigurable Devices', in *Field Programmable Logic and Application*, J. Becker, M. Platzner, and S. Vernalde, Eds. Springer Berlin Heidelberg, 2004, pp. 1032–1036.

[20] M. B. Stensgaard and J. Sparsø, 'ReNoC: A Network-on-Chip Architecture with Reconfigurable Topology', in *Second ACM/IEEE International Symposium on Networks-on-Chip (nocs 2008)*, 2008, pp. 55–64.

[21] N. Kapre and J. Gray, 'Hoplite: Building austere overlay NoCs for FPGAs', in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, 2015, pp. 1–8.

[22] X. Iturbe, K. Benkrid, T. Arslan, R. Torrego, and I. Martinez, 'Methods and Mechanisms for Hardware Multitasking: Executing and Synchronizing Fully Relocatable Hardware Tasks in Xilinx FPGAs', in *2011 International Conference on Field Programmable Logic and Applications (FPL)*, 2011, pp. 295–300.

[23] O. Sander, L. Braun, M. Hübner, and J. Becker, 'Data Reallocation by Exploiting FPGA Configuration Mechanisms', in *Reconfigurable Computing: Architectures, Tools and Applications*, 2008, pp. 312–317.

[24] M. Welter, 'Demonstration of Soft Error Mitigation IP and Partial Reconfiguration Capability on Monolithic Devices - XAPP1261 (v1.0)'. Xilinx Inc., 2015.

[25] J. Lamoureux and S. J. E. Wilton, 'FPGA Clock Network Architecture: Flexibility vs. Area and Power', in *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays*, New York, NY, USA, 2006, pp. 101–108.

[26] Xilinx Inc., 'UltraScale Architecture Clocking Resources, User Guide - UG572 (v1.8)'. Xilinx Inc., 19-Dec-2018.

[27] S. Verma and A. S. Dabare, 'Understanding clock domain crossing issues', *EE Times*, 2007.

[28] Xilinx Inc., 'Vivado Design Suite 7 Series FPGA and Zynq-7000 All Programmable SoC Libraries Guide - User Guide UG953 (v2016.2)'. Xilinx Inc., 2016.

[29] W. Simpson, 'PPP in HDLC-like Framing'. [Online]. Available: https://tools.ietf.org/html/rfc1662. [Accessed: 21-Jul-2016].

[30] S. Cheshire and M. Baker, 'Consistent overhead byte stuffing', *IEEEACM Trans. Netw.*, vol. 7, no. 2, pp. 159–172, Apr. 1999.

[31] P. Lin, 'One wire serial communication protocol method and circuit', US7111097B2, 19-Sep-2006.

[32] Xilinx Inc., 'Vivado Design Suite User Guide, High-Level Synthesis - UG902 (v2017.4)'. Xilinx Inc., 2018.

[33] V. Rantala, T. Lehtonen, and J. Plosila, 'Network on chip routing algorithms', Turku Centre for Computer Science, 2006.